METHOD AND APPARATUS FOR A LOW MEMORY HARDWARE IMPLEMENTATION OF THE KEY EXPANSION FUNCTION

The present invention relates to methods and apparatuses to perform encryption. More particularly, the present invention relates to the Cipher Key used by a Key Expansion algorithm to generate a set of Round keys in the Advanced Encryption Standard (AES).

With the increase in use of items such as Smartcards and commerce transacted over the Internet, the need to encrypt and decrypt data has never been more critical than in the present. In fact, the U.S. government, particularly through the National Institute of Standards and Technology (NIST) has for many years chosen encryption standards, such as DES (Data Encryption Standards) that was selected back in 1976 as the U.S. standard, and Triple DES subsequently became the standard. In recent years, the NIST has been evaluating a plurality of AES algorithms in order to select a new standard (AES) that would become the official encryption standard of the United States. Joan Daemen and Vincent Rijmen presented a cryptographic algorithm that has been approved by the NIST, and published on November 26, 2001 entitled Federal Information Processing Standards Publication No. 197 (hereinafter referred to as "FIPS'197). This algorithm in FIPS'197 is referred to as the Rijndael algorithm.

AES uses three systems of 128, 192 and 256 bits so as to improve the 56 bit encryption of the prior art in terms of performance, flexibility, efficiency, thereby providing an easier way to embody.

The basic unit for processing in the AES algorithm is a byte. Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the STATE.

The STATE generally has four rows of bytes, each containing Nb bytes, where Nb is the block length divided by 32.

Encryption converts data to an unintelligible form called cipher text. Decryption of the cipher text converts the data, which is referred to as "plaintext", back into its original form. Common terminology in the art refers to the series of transformations that converts plaintext to cipher text as "Cipher", whereas the series of transformations that converts cipher text to plaintext is referred to as "Inverse Cipher." In both Ciphering and Inverse Ciphering, a Cipher Key, which is a secret cryptographic key that is used by an Expansion Key Routine, generates a series of values (called round keys) that are applied to the STATE in the Cipher and Inverse Cipher routines.

The input and output for the AES algorithm each consists of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks, and the number of bits that they contain will be referred to as their length.

Fig. 1 illustrates an example of a STATE array input and output. At the start of the Cipher and Inverse Cipher, the input array of bytes (in_0 to in_{15}) is copied into the STATE as shown in Fig. 1. The Cipher or Inverse Cipher operations are then conducted on this STATE array, after which its final value is copied to the array of output bytes out_0 to out_{15}.
With regard to Fig. 1, as disclosed in "FIPS 197" by NIST, at the start of the Cipher and Inverse Cipher, the input, which is the array of bytes in_0 to in_{15}, is copied into the STATE array as shown. The Cipher or Inverse Cipher operations are then conducted on this State array, after which its final value is copied to the output, which is shown as the array of bytes ranging from out_0 to out_{15}. The addition of two elements in a finite field is achieved by "Adding" the

coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed with Boolean exclusive-or (XOR) operations ("FIPS '197", NIST, p. 10). Shown below is a binary notation example for adding two bytes:

$$\{01010111\} \oplus \{1000011) = \{11010100\}. \qquad \text{eqn (1.0)}$$

A Cipher Key is a secret, cryptographic key that is used by the Key Expansion algorithm to generate a set of Round Keys; it can be pictured as a rectangular array of bytes, having four rows and Nk columns ("FIPS '197", NIST, page 6).

The Round Keys are values that are derived from the Cipher Key by means of the key schedule. The key schedule includes two components: the Key Expansion and the Round Key Selection ("The Rijndael Cipher Block", Daemen and Rijmen, page 14). AES uses an algorithm that utilizes the Cipher Key, and generates a key schedule by performing the Key Expansion routine.

The Round Keys are applied to the STATE in the Cipher and Inverse Cipher transformations. The length of a Round Key is equal to the size of the STATE, and the Round Key is added to the STATE using an XOR operation.

Here we are concerned with reducing the size of the memory needed to support the Key Expansion. The current teaching provides for hardware implementation where the Cipher Key is expanded to support three key lengths: 128, 192 or 256 bits. The number of calculations depends on the KeyBits and blockbits. For 128 length keys, 10 calculations are required. For 192 length keys, 12 calculations are required. For the largest current key length (256), 14 calculations are required. Thus, the largest number of keys needed is 14.

The original AES software implementation uses storage for the maximum number of keys (14). In hardware, this storage requires 14 x 128 bit storage locations. An implementation of this feature in CMOS18 uses about 10,000 gates.

Thus, it would be advantageous to provide a method of decreasing the gate size and at the same time not increase the maximum path delay to any great extent, resulting in a smaller circuit that would be more attractive for high data-rate designs than known heretofore.

The present invention provides a method for decreasing the amount of gates required for processing the Expansion Key function without affecting the maximum path delay. An advantage of the invention is that the decreased amount of circuitry reduces the size, thus making the circuit smaller and more attractive for high data-rate designs.

According to an aspect of the present invention, each key expansion is performed to support both encryption and decryption. The new architecture utilizes a circuit that implements both normal and inverse key expansion that is synchronized with the Round Key processing and thus allows a smaller memory area to be used. In a particular embodiment, the memory space is decreased by one-half from the prior art 14 to the heretofore unknown number of 7 memory spaces for 14 expanded keys, with no increase in access time.

The above and other features and advantages of the present invention will become more apparent from the following detailed description when taken in conjunction with the accompanying drawings, in which:

FIG. 1 illustrates a STATE array and its input and output bytes.

FIG. 2 illustrates one way that an apparatus according to the present invention can be arranged.

FIG. 3 is a flow chart providing an overview a method according to the present invention.

In the following description, for purposes of explanation rather than limitation, specific details are set forth such as the particular architecture, interfaces, techniques, etc., in order to provide a thorough understanding of the present invention. However, it will be apparent to those skilled in the art that the present invention may be practiced in other embodiments, which depart from these specific details. Moreover, for the purpose of clarity, detailed descriptions of well-known devices, circuits, and methods are omitted so as not to obscure the description of the present invention with unnecessary detail.

Fig. 2 illustrates one way that an apparatus of the present invention can be arranged. A person of ordinary skill in the art understands that the illustrated physical layout is provided for explanatory purposes, and for example, the key expansion module and key scheduler can be combined, and the memory could be part of any module or a discrete area.

A keyschedule unit 201 has key values that are expanded by key expansion module 215. These values can be stored in memory 202. A conversion unit 208 encrypts/decrypts one of plaintext/ciphertext for a certain predetermined number of rounds, and a block round unit 212 carries out Round Key processing while the key expansion unit provides key expansion. Preferably, the key expansion function and Round Key processing is synchronized for each respective key so that these operation are performed at the same time (more or less in parallel).

According to the present invention, by implementing the key expansion algorithm with a smaller number of memory spaces, the timing of when the keys are used becomes an important factor. For example, during encryption mode, one key must be calculated before the process begins. While data is being processed by the Round Key algorithm, a second key can be calculated in parallel. If only a single encryption mode were needed, a single memory space would suffice.

The task is somewhat more difficult for decryption because key 14 is used first. Key 14 must be calculated based on the other 13 keys. The same idea of calculating keys in parallel with Round Key processing can still be used. If the order of writing keys into memory is carefully planned, both encryption and decryption can be supported with only seven memory spaces and very little additional control logic.

The rate at which keys are calculated relative to Round Key processing must also be carefully planned. By doing key calculations and Round Key processing in parallel with synchronization, the advantages of the present invention are realized.

The 256 key length requires the most memory so this length will be used to illustrate the new mechanism. Below is the conventional algorithm given in C code notation according to Rijndael (Rijndael-alg-ref.c version 2.0, Barreto and Rijmen) :

```
        Int rijndaelKeySched (word9 k[4][MAXKC], int keyBits, int blockBits, word8
        W [MAXROUNDS + 1] [4] [MAXABC])}
        /* Calculate the necessary round keys
         * The number of calculations depends on keyBits and blockBits
         */
    int KC, BC, ROUNDS;
    int I, j, t, rconpointer = 0;
    word8 tk[4] [MAXKC]
     switch (keyBits0 {
```

```
case 128: KC = 4; break;
case 192: KC = 6; break;
case 256: KC = 8;  break;
default : return (-1);
}
switch (blockBits) {
case 128: BC = 4; break;
case 192: BC = 6; break;
case 256: BC = 8;  break;
default : return (-2);
}
switch (keyBits >= blockBits ? keyBits : blockbits){
case 128: ROUNDS = 10; break;
case 192: ROUNDS =  12; break;
case 256: ROUNDS =  14; break;
default : return (-3); /* this cannot happen */
}
for (j =0; j< KC; j++)
   for (i = 0; i < 4; i++)
        tk [i] [j] = 1 [i] [j];
  t = 0;
 /* copy values into round key array */
```

```
            for (j = 0; (j < KC) && (t < (ROUNDS + 1) *BC); j++, t++)
                for (I = 0; i< 4; i++) W[t/BC] [i] [t % BC] = tk [i] [j];


            while (t < (ROUNDS + 1) * BC) {/* while not enough round key material
            calculated */
           /* calculate new values */
            for (I =0; I < 4; i++)
                     tk [i] [0] ^= S[tk[(i + 1) % 4] [KC - 1];
            tk [0] [0] ^ = rcon [rconpointer ++];
            if (KC ! =8)
               for (j = 1; j < KC; j++)
                    for (i = 0; i < 4; i ++) tk[i][j]  ^ = tk [i] [j-1];
             else {
                for (j = 1; j < KC/2; j++)
                     for (i=0; i < 4; i++) tk [i][j] ^ = tk  [i] [j-1]
                for (i = 0; i < 4; i++) tk [i][KC/2] ^ = S[tk [i]  [KC/2 - 1]];
                for (j = KC/2 + 1; j < KC; j++)
                         for (i = 0; i < 4; i++) tk [i] [j] ^= tk [i] [j - 1];
       }
       /* copy values into round key array */
       for (j = 0; (j < KC) && (t < (ROUNDS + 1) * BC); j++, t++)
              for (i=0; I < 4; i++) W[t/BC] [i] [t % BC] = tk [i] [j];
```

```
 }
  return 0;
}
```

It is noted that for a hardware implementation the maximum amount of memory for 14 keys must be allocated to implement the above code (case 256: ROUNDS = 14; break;).

It is also noted that this conventional expansion algorithm is done before the processing of Round Keys begins and does not allow for parallel activity.

In contrast to the above, the present invention uses a circuit that implements both normal and inverse key expansion, which is synchronized with the Round Key processing.

Look at encryption (normal) expansion, the example given will be the key length 256 as it requires the most memory locations. It follows that the other two cases (128 and 192 key lengths) also work because fewer Expanded keys are required. It also logically follows that if a key length is ever used that is greater than 256, the present invention is still effective in reducing required memory.

According to an aspect of the present invention, the algorithm for expanding keys and rounding keys in parallel is as follows:

```
Expand-Key_Number=14
For k=1 to Expand_Key_Number
{
if (k < 8) n=k else n = k-7
Expand key k and store to location n
Read location n and do Round Key algorithm, expand key k and store to location n+1
}
```

For the decryption (inverse) expansion all 14 keys must first be calculated. The calculations for the first seven keys are performed in descending order and then thrown away. The next seven keys are expanded in ascending location order. As the last seven keys are read in descending order, the first seven are re-expanded in descending ordering. Once key 8 is read from location 1, key seven is expanded and placed in location 1. The keys 7 through 1 are then read in ascending location order. The algorithm is as follows:

```
Expand_Key_Number = 14
For k =1 to Expand_Key_Number
{
 if (k < 8) n = 8 - k else n = k-7
 Expand key k and store to location n
}
For k = Expand_Key_Number to 1
{
  if (k < 8) n = 8 - k else n=k-7
 Read location n and do Round Key algorithm, expand key 8 -n and store to location n
}.
```

Finally, Fig. 3 shows the method steps explaining one way that the present invention can be performed. At step 305, there is provided a keyschedule unit 201 for a Cipher Key, said keyschedule unit 201 providing a predetermined number of expanded key values, and said keyschedule unit 201 having a memory 202.

Step 310 includes providing a conversion module 208 in communication with the keyschedule unit 201, the conversion module converts a block of plain text/ciphered text into a predetermined number of byte units in a first plurality of columns.

Step 315 includes providing a block round unit 212 for processing Round Keys for ~~encrypting/decrypting the predetermined number of byte units into ciphered text/plain text;~~ and

step 320 includes providing a key expansion module 215 for performing key expansion on both normal (encryption) and inverse (decryption) functions to obtain expanded key values;

wherein a rate at which the key values are expanded in step 320 is synchronized with Round Key processing by the block round unit 235 in step 315 so that each of the respective key values is expanded in parallel with a respective Round Key being processed.

Thus, the number of memory spaces in the memory 202 required for storage of the expanded key values is no greater than half the number of expanded key values.

By examining the design suggested by the AES proposal (NIST, November 26, 2001) implemented in two ways; a 128, 192 and 256 key length with 14 memory spaces and a second design with half the memory space and using careful recalculations in relation to the Round Key processing it has been shown by the present invention that the new algorithm will reduce the space required by one half.

It is understood by artisans of ordinary skill that there are various modifications that can be made that do not depart from the spirit of the invention or the scope of the appended claims. For example, the number of bits of common logic used, the layout of the modules and sub-modules of the apparatus, the number of blocks of data converted, the input and output modules, all can be modified according to need. As the present invention is capable of use with security networking processors, secure keyboard devices, magnetic card reader devices, smart card reader devices, and wireless communication applications such as 802.11 devices, the receipt or output of data can be contained within common circuitry or transmitted over RF, fiber optic, microwave, etc. In such cases a transmission and receive capabilities would be included, along with the protocol conversion from the various types of transmission. Further, while the examples show 8 bytes (128 bits), this amount could be increased or decreased according to need, and/or changes in the AES protocol. It should also be noted that terms such as "plain text" and "ciphered text" are terms of art and the encryption/decryption can encompass drawings, photos, illustrations, schematics, include voice, video, and/or multi-media data.

For example, the key expansion module is shown for illustrative purposes as a discrete unit that is separate form the keyscheduler, and the memory shown in the keyscheduler could be a common memory or arranged and/or all the components can be arranged on a common circuit card. While 128, 192 and 256 length key blocks were used for examples, any length key block larger or smaller can be used without departing from the spirit of the invention and the scope of the appended claims.